IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application for:

METHOD AND APPARATUS FOR ELIMINATION OF PROLOG AND EPILOG INSTRUCTIONS IN A VECTOR PROCESSOR

Inventors: Philip May, Silviu Chiricescu, Raymond Essick, Brian Lucas, Kent Moat, James Norris, Michael Schuette

Docket Number: CML00769D

METHOD AND APPARATUS FOR ELIMINATION OF PROLOG AND EPILOG INSTRUCTIONS IN A VECTOR PROCESSOR

CROSS REFERENCE TO RELATED APPLICATIONS

This application is continuation-in-part of related application serial number 10/184,772 (CML00108D) titled "Scheduler for Streaming Vector Processor", filed June 28, 2002 and application serial number 10/184,583 (CML00107D) titled "Reconfigurable Streaming Vector Processor", filed June 28, 2002.

FIELD OF THE INVENTION

This invention relates generally to the field of Vector Processors. More particularly, certain embodiments consistent with this invention relate to a method and apparatus for eliminating prolog and epilog programming instructions in a vector processor.

BACKGROUND OF THE INVENTION

Software pipelining for programmable, very long instruction word (VLIW) computers is a technique for introducing parallelism into machine computation of software loops. If different parts of the software loop use different hardware resources, the computation of one iteration of the loop may be started before the prior iteration has finished, thus reducing the total computation time. In this way several iterations of the loop may be in progress at any one time. In machines controlled by

VLIW instructions, the instructions in the middle of the loop (where the pipeline is full) are different from the instructions at the start of the loop (the prolog) and the instructions at the end of the loop (the epilog). If computation requires a number of different loops, a relatively large amount of memory is required to store instructions for the epilog and prolog portions of the loops.

Software pipelining for programmable VLIW machines, such as the IA-64, is accomplished by predicating instructions and executing them conditionally as the software pipeline fills and drains. The predication mechanism tags instructions with a predicate that conditions execution and committing of results to the register file in a general-purpose processor. This approach is generalized in these processors because the prediction mechanism is also used for general conditional execution. A disadvantage of this technique is the requirement for a centralized predicate register file.

OVERVIEW OF CERTAIN EMBODIMENTS OF THE INVENTION

Certain embodiments consistent with the present invention relate generally to a mechanism for eliminating prolog and epilog VLIW instructions in a vector processor without the use of centralized structures. Objects, advantages and features of the invention will become apparent to those of ordinary skill in the art upon consideration of the following detailed description of the invention.

In certain embodiments consistent with the invention, tags are associated with data elements in the pipeline to indicate the validity of the data. These tags are used to eliminate the prolog code, by suppressing the sinking of data tokens that are not tagged as valid. This approach avoids the use of centralized structures. The epilog instructions are eliminated by using sink-specific iteration counts, which again avoids the use of centralized structures.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, together with objects and advantages thereof, will best be understood by reference to the following detailed description of certain illustrative embodiments consistent with the present invention when read in conjunction with the accompanying drawing(s), wherein:

FIG. 1 is an exemplary schedule for a computation.

FIG. 2 is a loop body from the exemplary schedule.

FIG. 3 is an exemplary program schedule illustrating data tags and sink iteration counters of certain embodiments of the present invention.

FIG. 4 is an exemplary program schedule illustrating data tags and source iteration counters of certain embodiments of the present invention.

FIG. 5 is a diagrammatic representation of an embodiment of a re-configurable streaming vector processor.

FIG. 6 is an embodiment of an output stream unit of the present invention.

FIG. 7 is an embodiment of an input stream unit of the present invention.

FIG. 8 is an embodiment of a functional unit of the present invention.

FIG. 9 is flow chart illustrating an embodiment of a method of the invention.

FIG. 10 is flow chart illustrating the operation of a data source element in accordance with certain embodiments of the invention.

FIG. 11 is flow chart illustrating the operation of a data sink element in accordance with certain embodiments of the invention.

FIG. 12 is flow chart illustrating the operation of a functional element in accordance with certain embodiments of the invention.

DETAILED DESCRIPTION

While this invention is susceptible of embodiment in many different forms, there is shown in the drawings and will herein be described in detail one or more specific embodiments, with the understanding that the present disclosure is to be considered as exemplary of the principles of the invention and not intended to limit the invention to the specific embodiments shown and described. In the description below, like reference numerals are used to describe the same, similar or corresponding parts in the several Views of the drawings.

The Reconfigurable Streaming Vector Processor (RSVP) is a statically scheduled VLIW machine that executes dataflow graphs on vector data (data streams) in a highly pipelined fashion. The pipelining of calculations for the RSVP and other VLIW vector processors requires a significant amount of storage (memory) for prolog and epilog instructions. One aspect of the present invention is the elimination of the need to store prolog and epilog instructions by the use of two independent mechanisms: data tagging for prolog instruction elimination and sink-specific iteration counters for epilog instruction elimination.

The RSVP architecture provides resources for deeply pipelining computations, limited only by the true data dependencies and the resource limitations of the hardware. These deep computational pipelines would require many VLIW control words in the general case both for priming the pipeline (prolog) and draining the pipeline (epilog). In one example, the memory requirement for an MPEG4 encoder was reduced by 74% by eliminating prolog and epilog VLIW control words.

The prolog and epilog instructions are generally a subset of the instructions in the body of the loop. According to the present invention, mechanisms are provided to guard the execution (committing of results) to the correct subset of the loop-body. Consequently, the data path can repeatedly execute the loop-body, thereby eliminating the prolog and epilog code.

By way of example, a simple RSVP routine for performing a quantization operation will be described. The routine is coded in linearized flow form as:

```
        vbegin      7f-1f,0
1:      vld.s16     v1
2:      vsign.s16   1b
3:      vmul.s16    2b,s2
4:      vsub.s16    1b,3b
5:      vmul.s32    4b,s1
6:      vasru.s16   5b,16
7:      vst.s16     6b,v0
        vend
```

This exemplary linearized flow form uses the functional operations:

`vld.s16` -- load an element from a vector of 16-bit values

`vsign.s16` -- calculate the sign of an element from a vector of 16-bit values

`vmul.s16` --multiply two 16-bit vector elements

`vmul.s32` --multiply two 32-bit vector elements

`vsub.s16` --subtract two 16-bit vector elements

`vasru.s16` -- arithmetic shift right a 16-bit vector element

`vst.s16` -- store a 16-bit result vector element.

FIG. 1 shows how three iterations of the quantization loop can be scheduled. Referring to **FIG. 1**, each row denotes a time step and each column denotes a hardware functional unit (FU) or hardware resource of the vector processor. SRC denotes a data source or input and SINK denotes a data store or output. In this example, FU 1 is a logic device, FU 2 is a multiplier and FU 3 is an adder. At each time step a VLIW specifies the activity of each functional unit in addition to the location of the data used in the activity. The prefixed letters denote the iteration. For example, B3 is the third node of iteration B. In this example up to three iterations are being processed at one time (in time step 6 for example). The process is pipelined so that a new result is obtained every 3 cycles, whereas a single iteration requires 7 cycles. At the start and end of the computation, the pipeline is not filled, and the asterisks denote invalid data. In prior processors, the sequences of instructions involving invalid data are coded separately and are referred to as the prolog and the epilog. **FIG. 1** shows only three iterations of the loop. When multiple iterations are required, three instructions in the loop body are repeated, but the epilog and prologs are unchanged. The loop body is shown in **FIG. 2**. The loop body comprises three VLIW instructions. In this example, the prolog and epilog have 6 instructions each, while the loop body is only 3 instructions. Thus, the memory required to store the instructions for this computation is reduced significantly if the prolog and epilog are eliminated.

FIG. 3 illustrates one embodiment of the present invention. In accordance with a first aspect of the invention, data tokens in the functional units include both a data value and a data validity tag, or "valid bit". In a valid state, the data validity tag indicates that the data value is either the result of a valid token source's output, or the result of a calculation performed on all valid inputs. If this is the case, the token may be committed to memory (sunk). Otherwise, if neither condition is met, the data validity tag is in an invalid state and the data token is ignored at the sink, and is not committed to memory. Here, the term memory is taken to include any kind of data store, such as a memory cell or a register. The operation of the processor may be described by a dataflow graph and coded in linearized flow form. When execution of a graph is initiated, the intermediate result registers in the data path are all set to "invalid" so that only data from sources in the current execution of the graph will be committed. In addition, the data validity tags may be used to disable computations, thus saving energy. Referring to **FIG. 3**, the initialization is indicated by time step -1. The X's denote that data token in the corresponding functional unit or source unit is tagged as invalid. At subsequent time steps, the data tokens are replaced by valid tokens. At time step 0 input data is read and the SRC will contain a valid data token. At time step 2, this data token is passed to FU 1 and the operation A2 results in a valid data token. This process continues until the effective prolog is completed. In this manner, the VLIW instructions from the loop body may be used without alteration during the prolog, and the prolog instructions are eliminated. The data tokens propagate through the graph just as they do in the loop body, but the hardware recognizes when the data is invalid and tags the resulting output as invalid.

A second aspect of the present invention is the elimination of the epilog instructions. In one embodiment of this aspect, illustrated in **FIG. 3**, each token sink has its own copy of the iteration count for the currently executing graph. These iteration counts are decremented under VLIW control, and each sink is aware of its current iteration count. Individual iteration counts may differ due the deeply pipelined nature of the RSVP. This allows each sink to determine precisely when it needs to stop without referring to any centralized source. In the example shown in **FIG. 3**, a single sink is used. The sink iteration counter is initialized to three and is decremented at time steps 6, 9 and 12. At time step 3 the data is still tagged as invalid so the data is not sunk and the counter is not decremented. At time step 12 the last data token is sunk and the counter is decremented from 1 to 0. At time step 15 (and any subsequent time steps) the sink iteration counter has expired and the data token is again not sunk. If FU 1 were a sink, its sink counter would be decrement to zero at time step 11. Of course, counters may be incremented to a specified value rather than decremented, or may be decrement to a specified value other than zero. In this manner, the loop body code may be repeated for the epilog, and the epilog instructions are eliminated.

In addition, the input vector stream sources (input VSUs) may have private iteration counts to maintain the correct architectural state at graph completion (so that, architecturally, input addresses are left pointing to the next valid element in the array upon graph completion).

In a further embodiment of the present invention, source iteration counters are used for inputs to eliminate epilog instructions. This approach is illustrated in **FIG.**

4. Referring to the example shown in **FIG. 4**, which uses a single source, a source counter is initialized to 3 and is decremented at time steps 0, 3 and 6. At time step 9, the counter has expired, so the data token is marked as invalid (as indicated by the "X"). When the invalid data propagates through to the sink at time step 15, the tag is checked and the token is not sunk. In this approach, other data sources (including constant values stored in registers) may require counters.

An exemplary embodiment of RVSP hardware 100 is shown in **FIG. 5**. Referring to **FIG. 5**, the outputs and inputs of a number of functional units 102 are interconnected via a re-configurable interconnection switch 104. The functional units may include a multiplier 106, an adder 108, a logic unit 110 and a shifter 112. Other functional units may be included, and multiple functional units of a particular type may be included. The outputs from the functional units may be single registers or pipeline registers. The registers allow for storage of data tokens and the associated data validity tags. The functional units may support bit-slice operations. For example, the multiplier may have a 128-bit input and a 128-bit output and be able to perform two 32x32 to 64 or four 16x16 to 32-bit multiplies. In this case, a data validity tag is associated with each data value. The hardware also includes one or more accumulators 114. In the preferred embodiment, the accumulators act as both accumulators and storage registers, and are interfaced both to the interconnection switch 104 and to an external interface 116. The accumulators operate as data sinks and as functional elements. The external interface 116 enables the RSVP to link with a host processor and allows the host processor to access the accumulators and other parts of the RSVP. The functional units 102 and the re-configurable interconnection

switch 104 define the data-path for the RSVP. The functional units and the re-configurable interconnection switch 104 are linked to a controller 118 that includes a memory 120, preferably a cache, for storing a program of instructions describing the implementation specific description of a data-flow graph of the desired vector computation. At each cycle of the processor clock, the controller produces control words that configure the links in the interconnection switch and drive the function units. Storage and presentation of scalar values and tunnel node functionality is provided by constant unit portion of memory 120. The scalar values and tunnel initialization values may be loaded by the host processor or by the program of instructions. In operation, input data values are provided to the interconnection switch 104 by one or more input-stream units 122 (only one unit is shown the figure). Each input-stream unit 122 is controlled by a set of parameters describing the allocation of data in an external memory. This set of parameters is provided by the host processor, to which the input-stream unit is connected via external interface 116. The output-stream unit 124 is similarly controlled by the host processor and is operable to transfer data from the re-configurable interconnection switch 104 to external memory. The input-stream unit 122 and the output-stream unit 124 are linked to the controller 118 that synchronizes data flow.

An example of an output stream unit is shown in **FIG. 6**. Referring to **FIG. 6**, the output steam unit has a host interface, a memory interface, a control/status interface and a graph-side interface. In operation, the output stream unit is controlled by the control/status interface that is linked to the program controller. The "DONE" indicator signals that the output stream unit has produced its last valid token. The

DONE indicator may be asserted when the sink iteration counter of the output stream unit has expired. When all output stream units have signaled that they are finished, the controller knows that the computation has been completed. A "STALL" indicator is asserted if the output stream unit is not ready, as for example when the memory bandwidth is insufficient and the previous data from the graph side interface has not yet been written to memory. The graph-side interface receives data from functional units in the vector processor. In addition to the DATA itself, a data validity tag denoted by the signal VALID BIT is received which indicates if the data is valid. The VALID BIT signal is a digital signal representing one or more data validity tags. The VALID BIT signal is in an asserted state when the data is valid. In a further embodiment, the data comprises a number of sub-words, e.g. four 16-bit data values contained within a 64-bit data word. In this embodiment, four data validity tags (valid bits) are received, one for each sub-word. In the preferred embodiment, the logic for processing the sink iteration counter and for processing the data validity tags is implemented in hardware and an individual counter is associated with each output stream unit or other sink (such as an accumulator). Thus, the mechanism for epilog elimination is distributed.

An example of an input stream unit is shown in **FIG. 7**. Referring to **FIG. 7**, the input steam unit has a host interface, a memory interface, a control/status interface and graph-side interface. In operation, the input stream unit is controlled by the control/status interface that is linked to the program controller. The "DONE" indicator signals that the input stream unit has produced its last valid token. The DONE indicator may be asserted when the input device has read a specified number

of input tokens. A "STALL" indicator is asserted if the data from an external memory is not ready, as for example when the memory bandwidth is insufficient. The graph-side interface provides data to be passed to functional units in the vector processor. In addition to the DATA itself, a data validity tag, denoted by VALID BIT, is passed to the unit. The data validity tag is set if the data is valid. In a further embodiment, the data comprises a number of sub-words, e.g. four 16-bit data values contained within a 64-bit data word. In this embodiment, four data validity tags are used, one for each sub-word.

FIG. 8 is a diagrammatic representation of an exemplary functional unit of the present invention. Referring to **FIG. 8**, the functional unit receives one or more input operands (data tokens), indicated as OP A and OP B in the figure. Associated with input operand is a data validity tag that indicates whether the data token is valid. Data tokens may be invalid, for example, if they were produced as a result of operations on other invalid data tokens. The data may comprise a number of sub-words, e.g. four 16-bit data values contained within a 64-bit data word, in which case four VALID BITs are used, one for each sub-word. The functional units are controlled by CLOCK and ADVANCE signals and by instructions (OP CODES) derived from the VLIW instructions processed by the controller. The result of the operation performed in the functional unit is output, together with a corresponding number of VALID BITs. The logic for determining the output data validity tags from the input data validity tags is preferably implemented as hardware in the functional unit, so that no centralized resources are required.

One embodiment of the method of present invention is illustrated in the flow chart in **FIG. 9**. The flow chart illustrates a method of processing a single pipelined computation in a streaming vector processor. Following start block 200, all of the intermediate storage registers of the processor (e.g. the output registers of functional units) are initialized at block 202 so that the valid bit (data validity tag) of each register indicates that the data value is invalid. The data value need not be initialized, since the valid bit will ensure that it is not used in the subsequent calculation of a result. At block 204, the source and sink iteration counters are initialized. Typically, the initial values are equal to the loop count. At block 206 a step of the data graph (the program of instructions) is performed. This step is described in more detail below with reference to **FIGS. 10, 11** and **12**. At decision block 208, a check is made to determine if all of the data sinks have finished storing data values. As described below with reference to **FIG. 11**, each data sink indicates to the controller when is has finished storing results. If all the sinks are not yet finished, as indicated by the negative branch from decision block 208, flow returns to block 206. If all the sinks are done, as indicated by the positive branch from decision block 208, the computation is complete and terminates at block 210.

The flow chart in **FIG. 10** illustrates the method of operation of a source unit for a single program step in a pipelined computation. Following start block 220, a check is made, at decision block 222, to determine if the source iteration counter is equal to zero (expired). If the source iteration counter is expired, as indicated by the positive branch from decision block 222, flow continues to block 224, where the data validity tag associated with the output from the unit is set to indicate that the data is

invalid. If the counter has not expired, as indicated by the negative branch from decision block 222, the source iteration counter is decremented at block 226. Each source unit has a counter associated with it. A data value is retrieved from memory at block 228 and, at block 230, the data value is provided as an output. At block 232 the data validity tag associated with the data value is set to indicate that the value is valid and may be used in subsequent computations. The process terminates at block 234. Thus, the output data value is tagged with a data validity tag, which indicates the validity of the data value.

The flow chart in **FIG. 11** illustrates the method of operation of a sink unit for a single program step in a pipelined computation. Following start block 240, the incoming data value and the associated data validity tag are retrieved at block 242. At decision block 244 the validity of the data value is checked by examining the data validity tag. If the data is invalid, as indicated by the negative branch from decision block 244, the data value is not sunk to memory (block 246) and operation terminates at block 248. If the data is valid, as indicated by the positive branch from decision block 244, a check is made at decision block 250 to see if the sink iteration counter has expired. If it has expired, as indicated by the positive branch from decision block 250, a DONE signal is sent to the controller at block 252 to indicate that this source unit has completed its task for the current program loop. If the sink counter has not expired, as indicated by the negative branch from decision block 250, the sink iteration counter associated with this sink unit is decremented at block 254 and the data value is committed to memory at block 256. The process terminates at block 248.

It will be apparent to those of ordinary skill in the art that counters, such as the source iteration counter and the sink iteration counter, may be incremented or decremented. The term 'adjusting' will be used in the sequel to mean either incrementing or decrementing. In addition, the counters need not be initialized. Instead, the starting value may be noted and the difference between the starting value and current value used to determine if the counter has expired. Hence, counter initialization is taken to mean setting the counter to a specified value or noting the current value of the counter.

The flow chart in **FIG. 12** illustrates the method of operation of a functional (computational) unit of the vector processor. Following start block 260, the data values corresponding to the input operands are retrieved at block 262, together with the associated data validity tag for each data value. At decision block 264, the data validity tags are checked to determine if all data values to be used in the computation are valid. If not, as indicated by the negative branch from decision block 264, the data validity tag(s) associated with the output data value are set to indicate that the output data is invalid. If the input data is valid, as indicated by the positive branch from decision block 264, the functional operation is performed at block 270 and the output data value is calculated. At block 272, the data validity tag associated with the output data value is set to indicate that the data is valid and the process terminates at block 268.

Those of ordinary skill in the art will recognize that the present invention has been described in terms of exemplary embodiments based upon use of RSVP hardware architecture. However, the invention should not be so limited, since the

present invention could be implemented using other processors, which are equivalents to the invention as, described and claimed. Similarly, general purpose computers, microprocessor based computers, digital signal processors, microcontrollers, dedicated processors, custom circuits and/or ASICS may be used to construct alternative equivalent embodiments of the present invention. In addition, those skilled in the art will appreciate that the processes described above can be implemented in any number of variations and in many suitable programming languages without departing from the present invention. For example, the order of certain operations carried out can often be varied, additional operations can be added or operations can be deleted without departing from the invention. Such variations are contemplated and considered equivalent.

While the invention has been described in conjunction with specific embodiments, it is evident that many alternatives, modifications, permutations and variations will become apparent to those of ordinary skill in the art in light of the foregoing description. Accordingly, it is intended that the present invention embrace all such alternatives, modifications and variations as fall within the scope of the appended claims.